hakin9

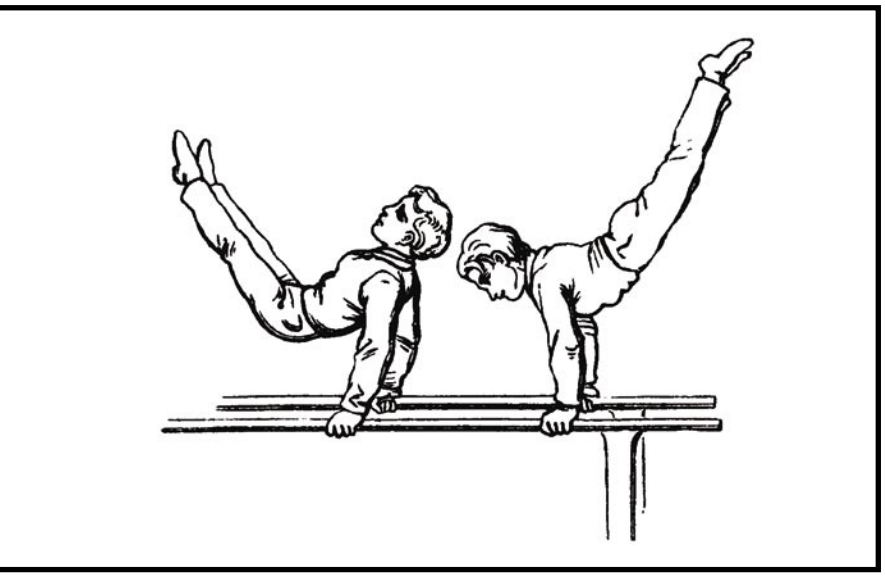
Instant Paranoia

Konstantin Klyagin

English version of an article published in "Hakin9" Magazine issue 3/2004

Instant Paranoia

Konstantin Klyagin



Talking every day on your favorite instant messaging networks with friends, lovers, colleagues and partners, have you ever thought if your conversations can be intercepted and become exposed to someone who can use them with some evil intentions?

There are many instant messaging systems with different features and capabilities. Their security properties are also different and it's obvious that one are more secure than others. We'll take a look at the major players on the IM market, see what vulnerabilities they have and using which of them you must be the most paranoid.

Basics

Imagine you came back from your job, sit down comfortably at your computer and open your favourite IM client. What happens then?

In a nutshell, when you login to the server, most client programs first open a socket connection to their main server. Through this connection a username (or UIN, depending on a network) and password pair is verified. If access is granted, the login procedure goes on proceeding to the list of contacts and offline messages that are held server-side until you login (if supported by network). Then you enter an idle state in which you can exchange messages, search for users and enjoy being in touch.

Sending and receiving messages is also done in more or less generic way for all the existing protocols. There are messages sent through the server (client -> server -> client), and those that are sent directly (client -> client, also known as *peer to peer*). Peer to peer is quite a common way of direct communications between two clients, though it is not supported in all of IM systems. These are used when one of the clients can connect to another one, knowing their IP address. Due to specific network setup or a firewall such connections are possible not to reach the destination. In this case, fallback to the through-server scheme is usually made.

What will you learn...

- what level of security and privacy do most popular IM protocols and applications provide,
- how an intruder can intercept passwords and communication sent via IM-s.

What should you know...

- sniffing basics (you may want to read articles *Sniffing for beginners* and *Sniffing in switched ethernet – Hakin9*, issues 1/2003 and 2/2003),
- perl basics (for understanding scripts).

How can one pass the traffic through one's computer

In order to sniff somebody's IM traffic one must first make this traffic pass through one's computer. This can be achieved in several ways:

- The simplest way is to run the sniffer on the very same machine on which the IM client runs. This way we can sniff our own IM talks to check if our IM client is secure or not. In all the following examples of sniffing we assume this is the case.
- One can also run the sniffer on any of computers through which the traffic passes. This can be the case of some malicious administrator who tries to spy users of his network.
- If an intruder has access to another machine in the same local network as the victim, sniffing traffic is also possible. It is easier if the network uses a hub, but it's also possible to sniff in a switched network using arp spoofing. For more information on sniffing read articles *Sniffing for beginners* and *Sniffing in switched ethernet* (*Hakin9*, issues 1/2003 and 2/2003). There is also a tutorial on arp spoofing available on our site (*http://www.hakin9.org*).

Threats

Obviously, the common vulnerability of almost all the instant messaging protocols is traffic being unencrypted. This means, no encryption is done on the data being sent and received, so a regular network sniffer can easily capture them. Besides messages, passwords are often sent as they are, which can be a real trouble. Anyone who has access to your network traffic can easily see your conversations and even steal your password.

Getting access to traffic is not simple. There is no way to sniff it remotely. In order to do that, a spy should get access to your computer, one of your gateways, or even one of computers on your LAN first. Any, any place your traffic goes through or nearby would suffice. Nearby, because there is also a technique called ARP spoofing that allows sniffing traffic even on networks with a switch (read more about it in the frame *How can you pass the traffic through your computer*). Programs such as Ettercap (*http://ettercap.sourceforge.net/*) can do that. A spy can also intrude into company network, or to that of the ISP your company is connected to. Even not a spy, but just a bored admin, having another sleepless night filled with cigars and caffeine is a potential privacy violator.

The easiest solution to protect network traffic including IM conversations is using protocols which use SSL – that eliminates the danger of occasional sniffing. But SSL isn't absolutely secure either, for it's vulnerable to attacks of the *man-in-the-middle* kind, especially in the conditions of lack of an appropriate PKI (public key infrastructure). The latter makes it possible to plug in your own public key into the connection and see the traffic as it is. Though, must say even a bare SSL is way more secure than plain connections, for it makes things more difficult for an intruder.

ICQ/AIM

The protocol of ICQ has all of the vulnerabilities mentioned above. Its latest version, OSCAR, establishes client-to-server and peer-to-peer connections without any traffic encryption mechanism. That makes it really trivial to sniff.

In order to check the security of protocols we will use tcpdump as the most common sniffer for UNIX-like systems further on. We assume you just use *Hakin9 Live*, so there's no need to explain how to install and configure it.

The command which we will use is:

```
# tcpdump -X -s 65535 \
  -i any 'port 5190 && tcp' | less
```

The parameters we use here mean that tcpdump should listen on every network interface in the system (`-i any`), produce mixed output of hex dump and printable ASCII representation of packets (`-X`) and display only packets sent from or to the local TCP port 5190 (`'port 5190 && tcp'`).

As soon as someone whose traffic goes through our machine sends a message, we see the output similar to presented on Listing 1.

Not only the message (which, as we can see, is *hi there*), but also destination UIN is transmitted in plain text and can be easily seen straight in the dump: it's 340274036.

Peer to peer connections

Sniffing peer-to-peer ICQ connections is a bit more difficult due to the fact that a random port is used for each session. When establishing a connection to the server, the client reports a port number it is going to use for peer-to-peer communications. Then each client that has us on his or her contact list receives the number of this port. They will use it to connect whenever there is a need to establish a direct connection. so it won't be a problem for a relatively advanced traffic analyzer application to catch the port number in the beginning of the session and treat this one too.

Listing 1. *What can we see when sniffing ICQ*

```
21:57:50.043968 our.hostname.32786 > 64.12.24.93.5190:
  P 1772083278:1772083343(65) ack 2250007272 win 36720 (DF)
0x0000   4500 0069 f0de 4000 4006 462a 5061 5abc        E..i..@.@.F*PaZ.
0x0010   400c 185d 8012 1446 699f d84e 861c 62e8        @..]...Fi..N..b.
0x0020   5018 8f70 5ad0 0000 2a02 3fb2 003b 0004        P..pZ...*.?..;..
0x0030   0006 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0001 0933 3430 3237 3430 3336 0002 0015        ...340274036....
0x0050   0501 0001 0101 0100 0c00 0000 0068 6920        .............hi.
0x0060   7468 6572 6500 0600 00                         there....
```

Listing 2. *Password sent in ICQ*

```
01:04:41.474121 our.hostname.39169 > bucp1-vip-m.blue.aol.com.5190:
  P 1:141(140) ack 11 win 5840 (DF)
0x0000   4500 00b4 6b22 4000 4006 425f 5061 5abc        E...k"@.@.B_PaZ.
0x0010   400c a199 9901 1446 a828 2c8d 0fb7 51d7        @......F.(,...Q.
0x0020   5018 16d0 788e 0000 2a01 051b 0086 0000        P...x...*.......
0x0030   0001 0001 0009 3334 3032 3734 3033 3600        ......340274036.
0x0040   0200 0891 53f5 b051 e3ba f600 0300 3349        ....S..Q......3I
0x0050   4351 2049 6e63 2e20 2d20 5072 6f64 7563        CQ.Inc..-.Produc
0x0060   7420 6f66 2049 4351 2028 544d 292e 3230        t.of.ICQ.(TM).20
0x0070   3030 622e 342e 3633 2e31 2e33 3237 392e        00b.4.63.1.3279.
0x0080   3835 0016 0002 010a 0017 0002 0004 0018        85..............
0x0090   0002 003f 0019 0002 0001 001a 0002 0ccf        ...?............
0x00a0   0014 0004 0000 0055 000f 0002 656e 000e        .......U....en..
0x00b0   0002 7573                                      ..us
```

Listing 3. *Finding the key used to encode passwords in ICQ*

```
#!/usr/bin/perl

$pass_orig = "butthead";
$pass_xored = "9153f5b051e3baf6";
$x = "";

for($i = $ix = 0; $i < length($pass_xored); $i += 2, $ix++) {
  $n = hex(substr($pass_xored, $i, 2)) ^ ord(substr($pass_orig, $ix, 1));
  $x .= chr($n);
}

for($i = 0; $i < length($x); $i++) {
printf "\\x%x", ord(substr($x, $i, 1));
}

print "\n";
```

Listing 4. *Example of a version of OSCAR protocol used by AOL Instant Messenger*

```
01:12:02.260926 our.hostname.1110 > 64.12.24.178.5190:
  P 1027:1155(128) ack 1859 win 8644 (DF)
0x0000   4500 00a8 e702 4000 8006 8366 c0a8 7680        E.....@....f..v.
0x0010   400c 18b2 0456 1446 0017 5be9 4bd8 2040        @....V.F..[.K..@
0x0020   5018 21c4 4a77 0000 2a02 4f80 007a 0004        P.!.Jw..*.O..z..
0x0030   0006 0000 557c 0006 3137 3841 4546 0000        ....U|..178AEF..
0x0040   0001 0974 6865 6b6f 6e73 7432 0002 0054        ...thekonst2...T
0x0050   0501 0003 0101 0201 0100 4900 0000 003c        ..........I....<
0x0060   4854 4d4c 3e3c 424f 4459 2042 4743 4f4c        HTML><BODY.BGCOL
0x0070   4f52 3d22 2366 6666 6666 6622 3e3c 464f        OR="#ffffff"><FO
0x0080   4e54 204c 414e 473d 2230 223e 7965 703c        NT.LANG="0">yep<
0x0090   2f46 4f4e 543e 3c2f 424f 4459 3e3c 2f48        /FONT></BODY></H
0x00a0   544d 4c3e 0003 0000                            TML>....
```

Due to the ICQ's immense popularity, even a lazy spy can easily find its protocol description on the Net. The most covering and well-structured unofficial specification can be found at *http://www.stud.uni-karlsruhe.de/~uck4/ICQ/*.

All of the existing guides are unofficial, because the protocol is proprietary and its specifications were never made public. So what we have is a reverse engineering result that is good enough to write a client or a traffic analyzer that will catch conversations and passwords.

Passwords

It's also worth mentioning that ICQ passwords are transmitted as xored strings, and it's very easy to find out position-dependent values they are xored with, by looking at a login packet that contains a password of the maximal length (8 characters). No need to change your password. Just set any UIN/password pair in the client and try to login. Watch what tcpdump says (Listing 2).

We know (we read it in above mentioned protocol specification) that encoded password starts at the fourth byte after the end of the UIN string. In our case it is `91 53f5 b051 e3ba f6`, we also know the original password – it was *butthead*. Since xor is a reversible operation (e. g. *a xor b=c* means that *c xor b=a*), knowing the original password and the xored result will do the trick. A simple Perl script (like the one presented at Listing 3) can be written to do that easily.

The result is `\xf3\x26\x81\xc4\x39\x86\xdb\x92`. That is the string that ICQ guys used.

IP for everyone

ICQ also exposes your IP address to the world. Those who have you on their contact list can see it. Authorization approval is not required, and this can easily be done with a simple packet.

Initially that was a very good intention to assure peer-to-peer communications. First versions of ICQ used to display the IP, then it was hidden from the user, which yet doesn't mean the information on IP address is not available. It is still transmitted by the server and less restrictive third party clients can kindly provide anyone with your IP address. You know what it means.

After the join

Since Mirabilis, the original ICQ creators, were bought by AOL, a major protocol join happened. The reason why I named this chapter ICQ/AIM is that the latest AOL Instant Messenger does also use a version of OSCAR protocol, despite some differences between them. First, packets with messages are built in a different way, but they are still unencrypted (Listing 4).

A small client-side digression

Speaking of clients, recently there are reports about worms and viruses aimed to exploit the standard client application. Like one day there was a message distributed all over the ICQ network inviting everyone to visit a web site with a funny cartoon. In fact, the page used a hole in Internet Explorer to get ICQ to send the deadly link further on to all of their contacts.

The solution here would be switching from the standard ICQ to Miranda, Trillian or other widespread instant messaging software. It mainly concerns Windows users of course.

Where UIN should be there is an AIM screen name. The major advantage of the AIM protocol against ICQ is that it is no longer possible to get someone's plain text password having access just to their traffic. The reason is that AIM uses the MD5 algorithm to make a hash of the password. It's a major step forward, since such hashes are generally irreversible. So hard times for a password cracker utility are guaranteed unless the password is something trivial. We can only guess why it is so only for AIM and for ICQ is not.

Finally, in spite of being easy to sniff, the ICQ protocol does have a tricky feature: its server listens on all the ports and it's able to provide the same protocol on any of them, from 1 to 65535. Just try telnet to *login.icq.com* yourself. This means, the most unintelligent sniffers (that are bound to a certain port) can be deceived by changing the default port number to something else in the client application.

AOL TOC

The old yet still used and recommended for third party client applications version of the AIM protocol is called TOC. This one maintains a single connection to *toc.oscar.aol.com:9898*.

Having adapted the *tcpdump* call for it we'll easily see messages on this one too:

```
# tcpdump -X -s 65535 \
  -i any 'port 9898 && tcp'
```

Which will give us the output as seen on Listing 5.

Nothing else to say here, besides the fact that passwords on TOC are also transferred xored. By applying the xor string finding technique described for ICQ, one can find out the phrase is *Tic/Toc*. Actually, there is no major secret about it, since back in 1998 the TOC protocol specification was released under the terms of GPL, and the phrase was explicitly given there. The specification was in the PROTOCOL file included into the distribution package of TiK, a Tcl/Tk client for AIM.

Listing 5. *Dump of TOC protocol*

```
22:24:19.037413 our.hostname.33010 > toc-m04.blue.aol.com.9898:
  P 1:49(48) ack 0 win 5840 (DF)
0x0000   4500 0058 7ecd 4000 4006 2cd3 5061 5abc        E..X~.@.@.,.PaZ.
0x0010   400c a3d6 80f2 26aa 1992 0d07 801d dc34        @.....&........4
0x0020   5018 16d0 921e 0000 2a02 dc46 002a 746f        P.......*..F.*to
0x0030   635f 7365 6e64 5f69 6d20 2274 6865 6b6f        c_send_im."theko
0x0040   6e73 7433 2220 2268 6920 7468 6572 6520        nst3"."hi.there.
0x0050   6f6e 2041 494d 2200                            on.AIM".
```

Listing 6. *Dump of TOC protocol -- xored password*

```
20:56:11.479212 our.hostname.34054 > toc-m08.blue.aol.com.9898:
  P 34:142(108) ack 11 win 5840 (DF)
0x0000   4500 0094 4198 4000 4006 69f3 5061 5abc        E...A.@.@.i.PaZ.
0x0010   400c a3af 8506 26aa bf02 d68a 8fe2 6018        @.....&.......`.
0x0020   5018 16d0 447b 0000 2a02 1644 0066 746f        P...D{..*..D.fto
0x0030   635f 7369 676e 6f6e 206c 6f67 696e 2e6f        c_signon.login.o
0x0040   7363 6172 2e61 6f6c 2e63 6f6d 2035 3139        scar.aol.com.519
0x0050   3020 2274 6f63 7669 6374 696d 2220 2230        0."tocvictim"."0
0x0060   7832 3730 3130 6335 6233 3331 6130 6422        x27010c5b331a0d"
0x0070   2065 6e67 6c69 7368 2022 6c69 6266 6972        .english."libfir
0x0080   6574 616c 6b20 7630 2e31 2e30 2d70 7265        etalk.v0.1.0-pre
0x0090   3230 2200                                      20".
```

Listing 7. *Perl script to obtain the TOC password in plain text*

```
#!/usr/bin/perl

$x = "Tic/Toc";
$pass_xored = "27010c5b331a0d";
$pass_orig = "";

for($i = $ix = 0; $i < length($pass_xored); $i += 2) {
  $n = hex(substr($pass_xored, $i, 2)) ^ ord(substr($x, $ix, 1));
  $pass_orig .= chr($n);
  $ix = 0 if ++$ix > length($x);
}
print "password = $pass_orig\n";
```

After xoring passwords are converted into a hex string we can see quoted right after the screen name (`0x27010c5b331a0d`). The following simple script in Perl (Listing 7) will obtain the password in plain text.

In rest, obviously, there is no protocol encryption or security layers either. So getting a temporary access to your network traffic will expose all of your terrible secrets. So don't ever tell anyone on the ICQ/AIM network where you buried the bodies.

Yahoo!

The very own messaging service of the first Internet portal in the world basically has the same security problem. There is no encryption,

Listing 8. *What tcpdump says about Yahoo!*

```
20:56:21.302176 our.hostname.42574 > cs47.msg.dcn.yahoo.com.5050:
  P 933410594:933410718(124) ack 3162461755 win 63712
  <nop,nop,timestamp 4138028 975496796> (DF)
0x0000   4500 00b0 2ab6 4000 4006 ca2a 5061 5abc        E...*.@.@..*PaZ.
0x0010   d89b c1ae a64e 13ba 37a2 b722 bc7f 563b        .....N..7.."..V;
0x0020   8018 f8e0 fa5f 0000 0101 080a 003f 242c        ....._.......?$,
0x0030   3a24 e65c 594d 5347 000b 0000 0068 0006        :$.\YMSG.....h..
0x0040   5a55 aa56 c57b 3985 31c0 8074 6865 6b6f        ZU.V.{9.1..sende
0x0050   6e73 74c0 8035 c080 7465 7374 6b6f 6e73        r12..5..destnick
0x0060   74c0 8031 34c0 8049 2062 7572 7269 6564        1..14..I.burried
0x0070   2074 6865 2062 6f64 6965 7320 696e 2074        .the.bodies.in.t
0x0080   6865 2062 6163 6b79 6172 642e 2044 6f6e        he.backyard..Don
0x0090   2774 2074 656c 6c20 616e 796f 6e65 21c0        't.tell.anyone!.
0x00a0   8036 33c0 803b 30c0 8036 34c0 8030 c080        .63..;0..64..0..
```

not even scrambling of the packets content. That is, knowing the port number would do the trick.

```
# tcpdump -X -s 65535 \
  -i any 'port 5050 && tcp'
```

This time our good buddy tcpdump says what's seen on Listing 8.

Here we have a packet that we can extract anything from. There are the both nicknames, one of the sender as well as destination's. And there is the message itself. However, *Yahoo!* protocol, just like AIM, uses MD5 to make a hash of the password, therefore password sniffing is much more difficult.

Yahoo! Messenger opens a single TCP/IP connection to its main server, *scs.msg.yahoo.com*, port 5050. Using it the authentication is done, as well as the text messaging that follows. However, there are separate servers for other services that the messenger provides. Catching file transfers would involve sniffing connections to *filetransfer.msg.yahoo.com:80* and *webcam.yahoo.com:5100* for webcam images respectively.

MSN

In the recent version of their protocol those of the great software monopoly decided to use SSL for all communications of their instant messaging service that currently goes under the name of .NET messenger. Only God knows what made them think for so long before applying this security measure – probably there was another lawsuit with some SSL making company.

Talking about connections that the client opens and closes during a work session, there is quite a handful of them in MSN. First it connects to the main server called *messenger.hotmail.com*, port 1863. No password check is done on this phase. Instead, the client is redirected to another, so called login server. Then follows the .NET passport (login and password pair) verification, which is done in two steps. First an HTTPS GET request to *nexus.passport.com* reads the hostname and port of the next server, which does the authentication (another HTTPS GET request).

Finally, when we are authorized, the client continues operating on the main TCP/IP connection until there is a need to initiate a conversation. Then the both clients open another connection to their server, so we get one stream per conversation, with a server between them, so it cannot really be considered peer-to-peer.

Even though everything is done through the server, with all of those redirections and a bunch of connections being opened and closed here and there, obtaining the original unencrypted traffic is the main difficulty in case of this very protocol. Once someone gets his hands on the traffic (by patching DNS and pretending a server, for example), there is no problem to get the rest. Needs to mention, there are still problems for an intruder, such as passwords encrypted the same way like they do it for Yahoo!, e. g. MD5 is used.

Making SSL obligatory is a really good idea, having on mind there is no guarantee that all the service users are security experts. As usual, MS addresses to regular computer users who are better to protect even without their consent. It would still be risky if there was an *enable SSL* checkbox which some well-wisher would recommend to uncheck, then getting direct to intercepting IM traffic of a naive user.

Just like the case with ICQ, there is an extensive unofficial documentation for the protocol, which is publicly available, located at *http://hypothetic.org/docs/msn/*.

Jabber

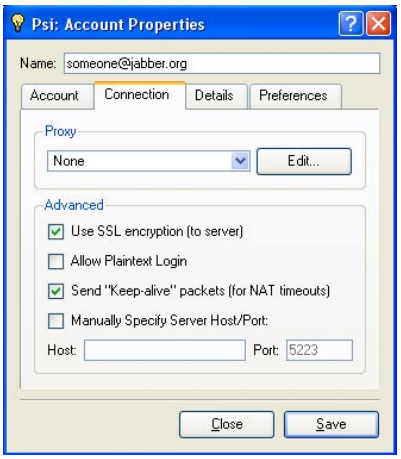
Skipping all yada yada about the protocol being flexible and thus, extremely handy for developers,

Keeping Jabber secure

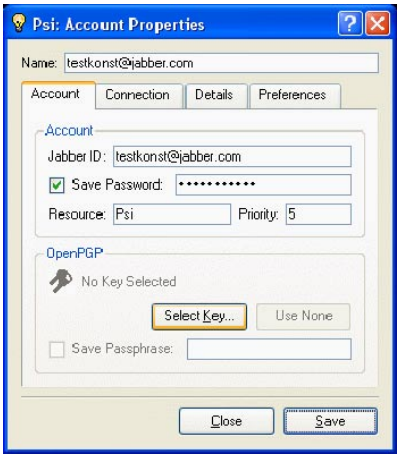
If you want to keep your Jabber messaging secure, here is an example on how to do it with the popular Psi client. Invoke the properties sheet for the account you are using, and check *Use SSL encryption* on the *Connection* tab. Also, make sure that the *Allow Plaintext Login* is not checked.

There is a possibility to make your Jabber even more secure by enabling the OpenPGP support and specifying your own PGP public key. In this case messages will be transmitted encrypted and only the recipient will be able to decrypt them, due to the private key being in his exclusive possession. However, one must remember that both sides must then use OpenPGP.

Not all client programs support that, but if we already chosen Psi, there is a way to do that. One must only install *gnupg*, then the key file can be set in the *Account* tab in the same account properties dialog.



Picture 1. *Use SSL encryption – secure Psi configuration*



Picture 2. *PGP key – secure Psi configuration*

freely available client and server implementations are also very flexible. So flexible that one can decide what level of security he wishes to have. The protocol uses a single TCP/IP connection and XML format for its packets. Catching the traffic would certainly do the trick, however the optional SSL support is among the standard features. For most clients a user has only to check the *use SSL* option in order to have all of his communications encrypted. Normally an SSL-enabled Jabber server would listen on two ports: 5222 for plain streams and 5223 for SSL.

The same rule applies to Jabber passwords too. It depends on the client if they are transmitted as a plain text or MD5 hash. This means that a properly configured client/server pair can provide a very reliable security measures where they are wanted.

Listing 9. *Jabber – this is how a message is sent*

```
<message type='chat' to='someone@server.org'>
  <body>hello</body>
</message>
```

Listing 10. *Jabber – this is how it looks when it reaches the destination client*

```
<message from='sender@server.org' to='someone@server.org' type='chat'>
  <body>hello</body>
</message>
```

Listing 11. *Jabber – example of an authentication packet with an MD5-hashed password*

```
<iq type='set' id='2'>
  <query xmlns='jabber:iq:auth'>
    <username>someone</username>
    <digest>554dfe01ecefb3d73e0c83f0c3f348b2378ce2c7</digest>
  </query>
</iq>
```

Besides client-to-server communications, the distributed nature of Jabber does also provide server-to-server connections that allow people using different servers to stay in touch, creating a whole decentralized network. The inter-server communication is established when some user tries to send an event to someone from another server. A major flaw here is that there is no option to make the server-to-server go through an SSL connection – it's always bare XML over TCP/IP.

So being a client and using SSL to connect to the server, it would mean that breaking into your network and just sniffing traffic wouldn't be enough for a spy to intercept your Jabber conversations. Instead, he must now think how to install a small implant into the network where your server is located. This will only give him a possibility to catch the traffic you send to users from other servers. Meanwhile all messaging within your server will remain secure.

Speaking of server-to-server communications in Jabber, it's worth mentioning a good method of protection against fake server connects it uses. The method is called *dialback* and it works like old international phone services – first you call the operator and tell them which phone abroad you want to dial, then they call you back and let you speak. Basically, here it's the same, though no humans are involved and DNS is used as a phone book.

The event-originating server establishes a TCP/IP connection to the receiving server through which it sends a packet containing some dialback key – a random text. Then the connection is closed. Now it's the receiving server's turn to lookup DNS for the originating server, make a connection and send the received dialback key for verification. Unless there is a hacked DNS in the server's network, this is a great yet simple protection against Jabber server spoofing.

Finally on Jabber, here are some packets illustrations. Due to its XML nature, all of them look very nice in plain text – Listing 9, 10 and 11.

The Human Factor

It's a well-known fact that the human is one of the weakest links in any computer system. Instant messaging is not an exception. So let's have a story before we finish.

Tools

Instant messaging is popular, no doubt about it. For many of us it's a usual way of everyday communication, just like phone, e-mail or crying out loud from the balcony to someone on the street. As the popularity of a particular IM service grows so does the amount of tools that allow to spy on users. These tools are mostly service-specific and much more complicated than our good traffic showing buddy, *tcpdump*. Well, their aim is different too.

There is a bunch of specific tools for ICQ, such as the shareware ICQ sniffer for Windows (*http://www.icq-sniffer.com/*) or spyware like Chat Watch (*http://www.zemericks.com/*), which, being installed on a victim's computer, allows to monitor their ICQ, MSN, Yahoo and AIM messages. Obviously, its slogan is *Protect your children from strangers*. Nothing is said about the age of such children – I guess it's ok for them to be 40+ years old.

The guys that created a Windows sniffer for ICQ have also got a solution for those who make a hard use of AIM. Guess what domain name they registered for it. Guessed? Right, it's *http://www.aimsniffer.com/*.

The latter does also have a namesake for UNIX, a project hosted at SourceForge, called *aimSniffer*, written in Perl using PCAP modules (*http://sourceforge.net/projects/aimsniff/*). It's able to store the results in an SQL database and even has a web front-end to access logs of conversations in a nice way.

For Gadu-Gadu there are some tools that allow to sniff conversations, for UNIX-like systems (*http://sourceforge.net/projects/ggsniff/*) as well as for Windows (*http://gg.wha.la/crypt/*).

Back to UNIX, a while ago I wrote a small tool whose aim was to demonstrate that on many IM networks (and not only) passwords are not really safe. The tool is called *kripp* (*http://thekonst.net/kripp/*), it is written in Perl, and it uses *tcpdump* through a pipe. For each service a separate *tcpdump* process is run with some specific parameters. There are also regexps defined for each service that are used to extract passwords from connections flow.

Using the script is simple. First, make sure you are *root* (remember, *tcpdump* runs only from the superuser account). If you run *kripp* without parameters, it starts watching traffic for all the supported networks, e.g. *icq*, *aim*, *ftp*, *http*, *cvs* and *pop3*. Optional parameters would include names of services passwords on which you would like to have logged. Here is an example of a session of kripp's work:

```
# kripp icq
  Protocols being kripped: icq
  icq password :: our.hostname -> ibucp-vip-m.blue.aol.com :: 123456789 :: butthead
```

The approach used in this simple tool can be applied to sniff conversations too. Using *tcpdump* gives the advantage of not having a need to get a clue with specific libraries, and flexible regexps of Perl make packets parsing easy.

Once upon a time there was a user. He used anti-virus monitors, installed only right and licensed programs, cleaned his tooth, brushed his hair, etc. Once a stranger knocked on his favorite instant messenger and they talked. The stranger gave many good computer-related advices. Among them there was a list of servers through which the instant messaging network could be accessed easier and faster. Well, the user went to the preferences and changed the server address to what the guy gave him and it worked well. In several days, the server went down. *Ok* – the user thought – *servers go down from time to time, so I'd better switch back to the default server.*

Table 1. *IM protocols from intruder's point of view – summary*

Protocol	*SSL*	*Password crypting*	*Other security measures*	*Other remarks*
ICQ OSCAR	No	XOR		
AOL OSCAR	No	MD5		
AOL TOC	No	XOR		
Yahoo!	No	MD5		
MSN	Yes	MD5		
Jabber	Yes	MD5	PGP	server to server communication in plain text

The truth is that the stranger was a spy. The server address he gave was that of his own computer, and he had a proxy that just redirected the traffic to the main server. This way, all the traffic going to and from the user was in his hands (classic *man-in-the-middle*). So as soon as the information he wanted to obtain went to his possession, there was no need to run the fake server anymore. And that's why it disappeared.

Scary enough, isn't it? But this is just like it is done when the technical part of the security measures taken is impeccable. Be sure this one is not the only way the human factor can be exploited. But instant messaging *is* for humans after all. And being a human is risky. Take care. ■